Docket: 043876-0154

PATENT

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of	:	Customer Number: 20277
HANSEN et al.	:	Confirmation Number: 4558
Application No.: 10/757,515	:	Group Art Unit: 2183
Filed: January 15, 2004	:	Examiner: Eric Coleman

For: METHOD AND SOFTWARE FOR MULTITHREADED PROCESSOR WITH PARTITIONED OPERATIONS

DECLARATION OF KORBIN VAN DYKE

I, Korbin Van Dyke, state that:

Personal Background

1. I am currently a sub-contractor of PatentVentures, an intellectual property consulting company with offices in San Jose, California and Austin, Texas. I have been a sub-contractor for or employed by PatentVentures as an intellectual property consultant since January 2002 and became a registered U.S. Patent Agent (Reg. No. 52,313) in August 2002. I am also owner/operator of Van Dyke Consulting (formerly known as Korbin S Van Dyke Consulting), since May 2004, an intellectual property and technical services consulting company with an office in Sunol, California.

2. I received a Master's of Science degree in Electrical Engineering and Computer Science (MS EECS) from the University of California (UC) Berkeley in 1982. Prior to receiving my Master's of Science degree, I received a Bachelor's of Science degree in Electrical Engineering and Computer Science from UC Berkeley in 1980.

3. While working toward my MS EECS degree, I also worked at UC Berkeley as a Research Assistant on the logic specification, logic verification, functional simulation, circuit design, and layout verification of an implementation of a Reduced Instruction Set Computer (RISC-I).

4. After graduating from UC Berkeley, I worked at various engineering and management positions from 1982 until 2001 that included involvement in the specification, implementation, and debugging of complex computer architecture systems. Further selected details of my work during this period are set forth below in paragraphs 5 through 8.

5. From June 1982 to October 1987, I worked at VLSI Technology in San Jose, California as a VLSI Systems Engineer. I specified and designed a memory interface for a multi-mode display interface chip. I defined the instruction set architecture and microarchitecture of a programmable digital signal processor (DSP), and led a team that built and evaluated a microprocessor implementation of the DSP.

6. From October 1987 to May 1996, I worked at Nexgen Microsystems and remained with the company after it was acquired by Advanced Micro Devices (AMD) in January 1996, in various offices in San Jose and Milpitas, California. I designed the microarchitecture and logic details of the pipeline control for a superscalar, out-of-order, x86-compatible, multi-chip microprocessor, the Nx586. I also micro-architected and logic designed the branch prediction and multiple stream instruction fetch hardware of the Nx586. I oversaw various aspects of the physical design of the Nx586 (and its predecessors), including synthesis from Verilog code, place and route, timing closure, and layout versus schematic/netlist verification. I led a team that verified architectural correctness of the Nx586, and I led a team that debugged the Nx586 multi-chip silicon. I investigated several micro-architectural tradeoffs with respect to future high-performance x86 processors.

7. From May 1996 to May 2000, I worked at Chromatic Research and remained with the company after it was acquired by ATI Research Silicon Valley in November 1998, in Sunnyvale, and then Santa Clara, California. I oversaw the development and made personal contributions to the instruction set architecture of a dual-instruction-set processor (x86-compatible and RISC), including specifications for efficient transitions between the instruction sets, architectural hooks for efficient dynamic binary translation from the x86 instruction set to the RISC instruction set, and SIMD arithmetic and special hardware assists for media processing operations. I led a team that developed the pipeline control section of the processor (including Verilog coding and synthesis), and oversaw a team that developed the physical design of portions of the processor (place and route).

8. From June 2000 to December 2001, I worked as the Director of VLSI Development at XStream Logic and remained with the company as an Architect (after the company changed its name to Clearwater Networks in October 2001) in Los Gatos, California. As a VLSI Director, I led a team that was responsible for implementing a complex network processor from a set of specifications to silicon, using a simulatable high-level language description of the processor as a starting point. The team used a tool set that enabled automatic conversion of the high-level language into Verilog for simulation and synthesis, and then placement and routing. As an Architect, I defined and documented various architectural approaches for future network processors.

9. My area of specialization is architecture, microarchitecture, and logic design of complex VLSI systems, particularly microprocessors. I am a co-inventor on more than 50 issued U.S. patents, relating to a variety of subjects, including multiple ISA execution, hardware

optimization for binary translation, video decoding ISA optimization, compatible ISA implementation techniques, speculative instruction execution, branch prediction, compatible segmentation and paging, standard PC sub-system virtualization, address generation, and logarithmic calculation.

10. A copy of my CV is attached as Exhibit A.

Summary of My Opinions

11. In preparation of this declaration I have reviewed U.S. Patent Application Serial No. 10/757,515 including the specification and the appendix referenced therein (the Zeus System Architecture Manual – hereinafter referred to as the "Zeus manual"). I have also reviewed the "Office communication" for the 10/757,515 patent application mailed on January 29, 2007 including the two paragraphs on pages 2-3 that discuss the Rejection of claims 1-9 under 35 U.S.C. § 112, first paragraph, as failing to comply with the written description requirement and the enablement requirement. My understanding of the written description requirement is that a patent disclosure must describe the claimed invention in sufficient detail that one of ordinary skill in the art can reasonably conclude that the inventor had possession of the claimed invention at the time of filing the patent disclosure. My understanding of the enablement requirement is that the patent disclosure must contain sufficient information regarding the subject matter of the claims to enable one of ordinary skill in the pertinent art to make and use the claimed invention. I further understand that whether the enablement requirement is met depends on whether undue experimentation is necessary for one of skill in the art to practice the invention in light of the patent disclosure.

12. Based on my review of the materials identified in paragraph 11, it is my opinion that:

> (i) the disclosure of the 10/757,515 patent application indicates that the inventors were in possession of the claimed (as amended) "executing instructions from each instruction stream received at the execution unit in a multistage pipeline such that, at a given time, the multistage pipeline includes instructions from different ones of the instruction streams in different stages of the multistage pipeline" of the 10/757,515 patent application; and
>
> (ii) the disclosure of the 10/757,515 patent application would have enabled a person of ordinary skill in the art to make and use, without undue experimentation, the claimed (as amended) "executing instructions from each instruction stream received at the execution unit in a multistage pipeline such that, at a given time, the multistage pipeline includes instructions from different ones of the instruction streams in different stages of the multistage pipeline" of the 10/757,515 patent application.

13. The disclosure of the 10/757,515 patent application provides detailed information and description that I believe indicates that the inventors were in possession of the claimed (as

amended) "executing instructions from each instruction stream received at the execution unit in a multistage pipeline such that, at a given time, the multistage pipeline includes instructions from different ones of the instruction streams in different stages of the multistage pipeline", and that I further believe would have enabled a person of ordinary skill in the art to make and use the claimed invention without undue experimentation. On at least pages 49-53 (the Pipeline Organization section) of the Zeus manual (attached as Exhibit B) there are detailed descriptions of various pipeline organizations and operational techniques.

14. A detailed explanation of the basis for my opinions is set forth in the remainder of this declaration.

Detailed Basis for My Opinions

Level of Ordinary Skill in the Art:

15. Based on my review of the 10/757,515 patent application (referred to as the "media processor patent application"), I believe that the media processor patent application pertains to the technology of microprocessor design and microprocessor systems.

16. In my opinion a person of ordinary skill in the art of microprocessor design and microprocessor systems would possess a bachelor's degree in electrical engineering or computer science and have approximately five years of experience in the field of microprocessor design and microprocessor systems. Alternatively, an equivalent person could have had a master's degree in electrical engineering or computer science and have approximately two or more years of experience in the field of microprocessor design and microprocessor systems. This person would readily understand the conceptual design of a microprocessor and a microprocessor execution unit. The person of ordinary skill would have had access to a library of technical publications, periodicals, and textbooks.

Analysis of the disclosure of the 10/757,515 patent application:

The disclosure of the 10/757,515 patent application describes the claimed "executing instructions from each instruction stream received at the execution unit in a multistage pipeline such that, at a given time, the multistage pipeline includes instructions from different ones of the instruction streams in different stages of the multistage pipeline" in sufficient detail that a person of ordinary skill in the art could reasonably conclude the inventors were in possession of the claimed invention, and that a person of ordinary skill in the art would have been enabled to make and use the claimed invention without undue experimentation:

17. Claim 1 (as amended) of the 10/757,515 patent application recites, among other elements: "executing from each instruction stream received at the execution unit in a multistage

pipeline such that, at a given time, the multistage pipeline includes instructions from different ones of the instruction streams in different stages of the multistage pipeline".

18. The claimed invention includes/performs three different features and functions. Namely, the executing is characterized by: (1) a multistage pipeline; (2) executing from each instruction stream received at the execution unit in the multistage pipeline; and (3) the multistage pipeline includes, at a given time, instructions from different ones of the instruction streams in different stages of the multistage pipeline. I believe the disclosure of the 10/757,515 patent application describes each of these three features/functions in sufficient detail so that a person of ordinary skill in the art could reasonably conclude that the inventors had possession of the claimed invention. I further believe the disclosure of the 10/757,515 patent application describes each of these three features/functions in sufficient detail so that a person of ordinary skill in the art would have been enabled to make and use the claimed invention without undue experimentation. Details of where in the disclosure of the 10/757,515 patent application each of these three claim elements are set forth is discussed below.

(1) a multistage pipeline: The 10/757,515 patent application discloses pipelines that, in some cases, require several stages to complete execution; i.e. the pipelines are "multistage". For example: "dependent operations must be separated by the latency of the pipeline, and for highly pipelined machines, the latency of simple operations can be quite significant" (10/757,515 patent application, paragraph [0138]). Based on this disclosure, I believe a person of ordinary skill in the art would readily conclude that the 10/757,515 patent application describes a multistage pipeline.

(2) executing from each instruction stream received at the execution unit in the multistage pipeline: The 10/757,515 patent application discloses sharing execution units among threads, with the execution units performing operations from various threads. For example: "Instructions for execution units, which are shared to varying degrees among the threads are also buffered for later execution. The execution units then perform operations from all active threads using functional data path units that are shared." (10/757,515 patent application, paragraph [0134]). Thus, I believe a person of ordinary skill in the art would readily conclude that the disclosure of the 10/757,515 patent application describes executing from each instruction stream received at the execution unit in the multistage pipeline.

(3) the multistage pipeline includes, at a given time, instructions from different ones of the instruction streams in different stages of the multistage pipeline: The Zeus manual clearly describes pipelined processing (in a multistage pipeline) of instructions of different threads (instruction streams). For example: "A Zeus Superthread pipeline, with 5 simultaneous threads of execution, permits simple operations, such as register-to-register add (G.ADD), to take 5 cycles to complete, allowing for an extremely deeply pipelined implementation." (Zeus manual, p 53). Processing of the different instructions overlaps in time in the stages of the multistage pipeline, leading to instructions of the different streams being in different stages of the multistage pipeline at various overlapping times. Thus, I believe a person of ordinary skill in the art would readily conclude that the disclosure of the 10/757,515 patent application describes that the multistage pipeline includes, at a given time, instructions from different ones of the

instruction streams in different stages of the multistage pipeline. Further details relating to this sub-paragraph, as well as sub-paragraphs (1) through (2) above, are set forth below in paragraphs 21 through 26.

19. Based on the above, I believe that a person of ordinary skill in the art would readily conclude that the disclosure of the 10/757,515 patent application describes "executing from each instruction stream received at the execution unit in a multistage pipeline such that, at a given time, the multistage pipeline includes instructions from different ones of the instruction streams in different stages of the multistage pipeline", as recited in claim 1 (as amended) of the 10/757,515 patent application, in sufficient detail to conclude that the inventors had possession of the claimed invention, and that the disclosure of the 10/757,515 patent application provides sufficient detail to enable a person of ordinary skill in the art to make and use the claimed invention without undue experimentation.

In addition to describing "executing instructions from each instruction stream received at the execution unit in a multistage pipeline such that, at a given time, the multistage pipeline includes instructions from different ones of the instruction streams in different stages of the multistage pipeline", as recited in claim 1 (as amended), the disclosure of the 10/757,515 patent application also describes the remaining elements in sufficient detail that a person of ordinary skill in the art could reasonably conclude the inventors were in possession of the claimed invention, and that a person of ordinary skill in the art would have been enabled to make and use the claimed invention without undue experimentation:

20. Claim 1 (as amended) of the 10/757,515 patent application recites additional elements other than "executing instructions from each instruction stream received at the execution unit in a multistage pipeline such that, at a given time, the multistage pipeline includes instructions from different ones of the instruction streams in different stages of the multistage pipeline". Namely, the additional elements are: (1) "storing a plurality of data elements in partitioned fields of at least one register having a register width, each of the data elements having an elemental width smaller than the register width"; (2) "receiving an instruction stream for each one of the plurality of threads at an execution unit"; and (3) "the instructions including a single instruction that specifies an operation to cause multiple instances of the operation to be performed, each instance of the operation to be performed using a different one of the plurality of data elements in partitioned fields of the at least one register to produce a catenated result". I believe the disclosure of the 10/757,515 patent application describes each of these three functions/features in sufficient detail so that a person of ordinary skill in the art could reasonably conclude that the inventors had possession of the claimed invention. I further believe the disclosure of the 10/757,515 patent application describes each of these three features/functions in sufficient detail so that a person of ordinary skill in the art would have been enabled to make and use the claimed invention without undue experimentation. Selected details of where in the disclosure of the 10/757,515 patent application each of these three claim elements are set forth is discussed below.

(1) "storing a plurality of data elements in partitioned fields of at least one register having a register width, each of the data elements having an elemental width smaller than the register width" – the 10/757,515 patent application discloses, for example, execution registers, such as ER 125, coupled to units E 141, X 142, and G 143 (see Fig. 1 of the

10/757,515 patent application), that are 128 bits wide (10/757,515 patent application, paragraph [0106]). As another example, the various group and ensemble operations described in the Zeus manual clearly indicate that elemental data widths smaller than the register width are used. See, for instance, the "group add" operations such as "G.ADD.8", "G.ADD.16", "G.ADD.32", and G.ADD.64" (Zeus manual, pp. 135-137, attached as Exhibit C). See, for another instance, the "ensemble floating-point" operations such as "E.ADD.F.16", "E.ADD.F.32", and "E.ADD.F.64" (Zeus manual, pp. 258-260, attached as Exhibit D). In both instances the integer suffixes, "8", "16", "32", and "64", respectively, indicate a size of partitions of bits of the operands, and all of the sizes are smaller than the 128-bit wide registers. Based on this disclosure, I believe a person of ordinary skill in the art would readily conclude that the disclosure of the 10/757,515 patent application describes "storing a plurality of data elements in partitioned fields of at least one register having a register width, each of the data elements having an elemental width smaller than the register width" in sufficient detail that a person of ordinary skill in the art would have reasonably concluded that the inventors had possession of the claimed invention, and that a person of ordinary skill in the art would have been enabled to make and use the claimed invention without undue experimentation.

(2) "receiving an instruction stream for each one of the plurality of threads at an execution unit" -- the 10/757,515 patent application discloses, for example, presentation of instructions from four threads to execution units: "instructions are coupled to the execution unit arbitration unit Arbitration 131, that selects which instructions from the four threads are to be routed to the available execution functional units E 141 and 149, X 142 and 148, G 143-144 and 146-147, and T 145" (10/757,515 patent application, paragraph [0079]). As another example, the Zeus manual includes a diagram showing "thread number which issues an instruction" versus "functional units [that] may be used by that instruction", indicating receipt of instructions of various threads (instruction streams) by various parts of execution units (Zeus manual, p. 53, see Exhibit B). Based on this disclosure, I believe a person of ordinary skill in the art would readily conclude that the disclosure of the 10/757,515 patent application describes "receiving an instruction stream for each one of the plurality of threads at an execution unit" in sufficient detail that a person of ordinary skill in the art would have reasonably concluded that the inventors had possession of the claimed invention, and that a person of ordinary skill in the art would have been enabled to make and use the claimed invention without undue experimentation.

(3) "the instructions including a single instruction that specifies an operation to cause multiple instances of the operation to be performed, each instance of the operation to be performed using a different one of the plurality of data elements in partitioned fields of the at least one register to produce a catenated result " -- the 10/757,515 patent application discloses, for example, "execution functional units G 143-144 and 146-147 [that] are group arithmetic and logical units that perform simple arithmetic and logical instructions, including group operations wherein the source and result operands represent a group of values of a specified symbol size, which are partitioned and operated on separately, with results catenated together" (10/757,515 patent application, paragraph [0080]). As another example, the Zeus manual describes various group and ensemble

operations that operate on partitions of bits of registers and catenate results in another register. The group and ensemble operations include group integer arithmetic and Boolean operations, as well as group floating point arithmetic operations. See, for instance, "group add" operations that "take operands from two registers, perform operations on partitions of bits in the operands, and place the catenated results in a third register" (Zeus manual, pp. 135-137). Each of the group operations performs the same operation on partitions of bits in the operands; e.g. the "group add" operation performs an "add" operation on partitions of bits in the operands, and results are catenated together. See, for another instance, "ensemble floating-point" operations that "take two values from registers, perform a group of floating-point arithmetic operations on partitions of bits in the operands, and place the catenated results in a register" (Zeus manual, pp. 258-260). Each of the ensemble floating-point operations performs the same operation on partitions of bits in the operands; e.g. the "ensemble add floating-point" operations perform an "add floating-point" operation on partitions of bits in the operands, and results are catenated together. Based on this disclosure, I believe a person of ordinary skill in the art would readily conclude that the disclosure of the 10/757,515 patent application describes "the instructions including a single instruction that specifies an operation to cause multiple instances of the operation to be performed, each instance of the operation to be performed using a different one of the plurality of data elements in partitioned fields of the at least one register to produce a catenated result " in sufficient detail that a person of ordinary skill in the art would have reasonably concluded that the inventors had possession of the claimed invention, and that a person of ordinary skill in the art would have been enabled to make and use the claimed invention without undue experimentation.

Thus the 10/757,515 patent application discloses each and every element of claim 1 (as amended) in sufficient detail such that a person of ordinary skill in the art could reasonably conclude the inventors were in possession of the claimed invention, and that a person of ordinary skill in the art would have been enabled to make and use the claimed invention without undue experimentation.

Detailed analysis of the disclosure of the 10/757,515 patent application with respect to "executing instructions from each instruction stream received at the execution unit in a multistage pipeline such that, at a given time, the multistage pipeline includes instructions from different ones of the instruction streams in different stages of the multistage pipeline":

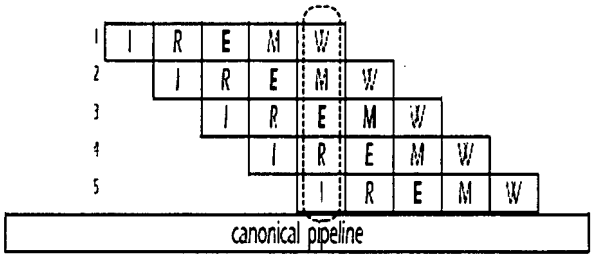
21. I have reviewed the 10/757,515 patent application disclosure, including the portions of the Zeus System Architectural manual that relate to the organization and operation of pipelined hardware structures such as canonical, superspring, and superthread pipelines. The 10/757,515 patent application provides several examples of pipeline organization and operation that would be readily understood by one of ordinary skill in the art. The examples are detailed and described with clarity and precision that would facilitate implementation without undue experimentation. Thus, I believe the 10/757,515 patent application includes sufficient detail that

a person of ordinary skill in the art would have reasonably concluded that the inventors had possession of, and would have been enabled to make and use, without undue experimentation, "executing instructions from each instruction stream received at the execution unit in a multistage pipeline such that, at a given time, the multistage pipeline includes instructions from different ones of the instruction streams in different stages of the multistage pipeline" as recited in claim 1 (as amended) of the 10/757,515 patent application.

22. Several example pipeline organizations described by the 10/757,515 patent application disclosure follow. The example pipeline organizations illustrate instances of "executing instructions from each instruction stream received at the execution unit in a multistage pipeline such that, at a given time, the multistage pipeline includes instructions from different ones of the instruction streams in different stages of the multistage pipeline".

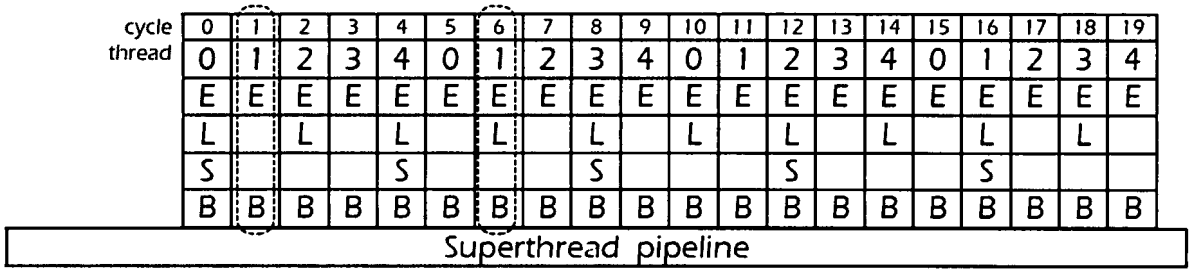
23. A first example pipeline organization is with respect to a canonical pipeline, in conjunction with a Zeus Superthread pipeline.

(a) An exemplary canonical pipeline is illustrated in a figure on p. 49 of the Zeus manual, and reproduced with annotation directly below:



As described in the Zeus manual, the foregoing figure illustrates a five-stage pipeline (stages I, R, E, M, and W) operating for five cycles. The illustrated pipeline is an example of "a multistage" pipeline (since there are five stages). Also note that, for instance, in the encircled fifth cycle, there are five instructions in the pipeline, with each instruction being in a respective one of the five pipeline stages.

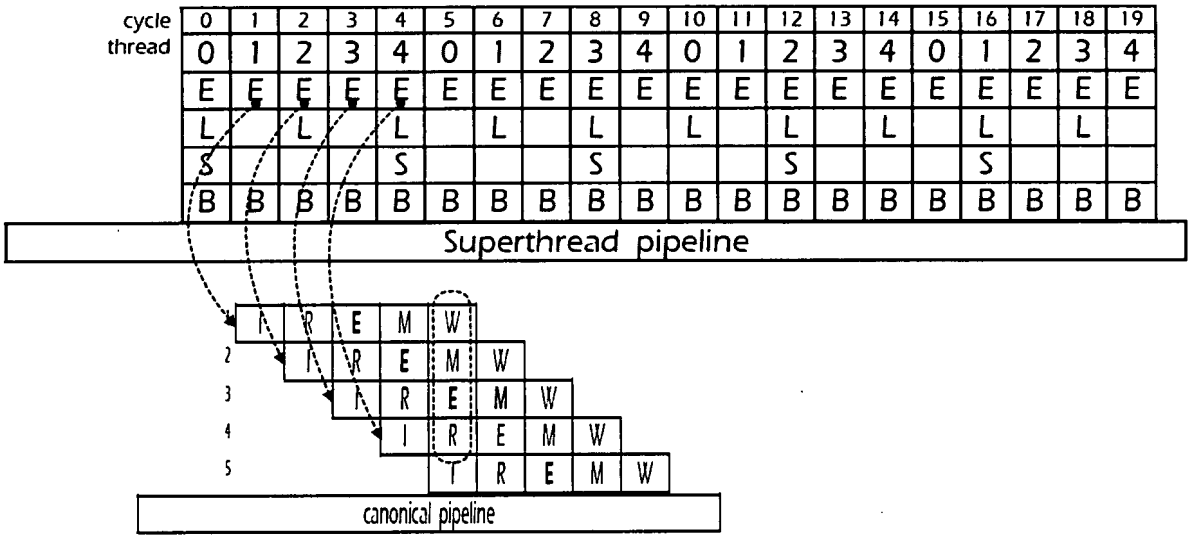
(b) An exemplary Zeus Superthread pipeline is illustrated in a figure on p. 53 of the Zeus manual, and reproduced with annotation directly below:



As described in the Zeus manual, the foregoing figure illustrates issue of instructions according to five threads: "In the diagram below, the thread number which issues an instruction is indicated on each clock cycle" (Zeus manual, p. 53); thread numbers

shown are 0, 1, 2, 3, and 4, for a total of five threads. Note that, for instance, in the encircled first cycle, the first thread issues an instruction using any combination of the E and B resources; in the encircled sixth cycle, the first thread issues an instruction using any combination of the E, L, and B resources.

(c) One of ordinary skill in the art would have understood the canonical pipeline stages as operated in accordance with the Zeus Superthread pipeline as the following shows. Consider the following figure with annotations superimposed on the foregoing Zeus Superthread and canonical pipeline illustrations:

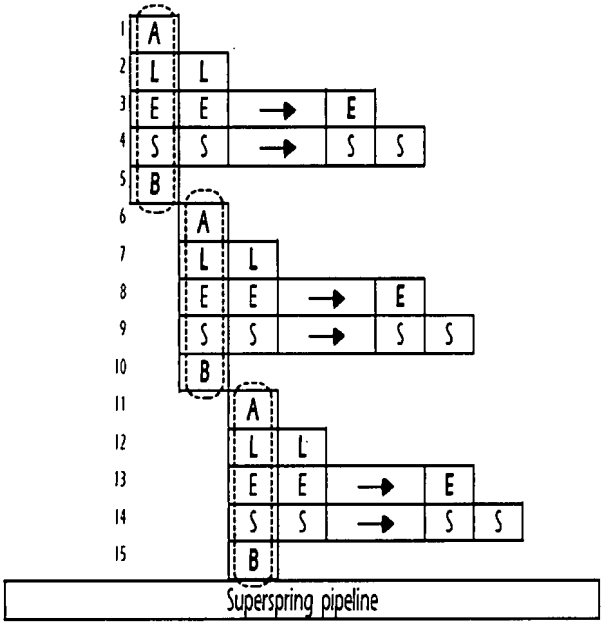


On the first cycle, the first thread of the Zeus Superthread pipeline issues a first instruction that uses the E resource. The annotation shows the first instruction as it would be processed by the canonical pipeline when operated in accordance with the Zeus Superthread pipeline. Similar annotations illustrate the second, the third, and the fourth threads issuing respective second, third, and fourth instructions. Note that, for instance, in the encircled fifth cycle of the canonical pipeline stages, the first through fourth instructions are in the canonical pipeline, in respective ones of the canonical pipeline stages (the W, M, E, and R stages), and each of the first through fourth instructions are respectively from the first through fourth threads. Thus on at least the fifth cycle, the canonical pipeline is an example of "[a] multistage pipeline [that] includes instructions from different ones of the instruction streams in different stages of the multistage pipeline".

(d) Thus the first example shows how the '10/757,515 patent application disclosure describes the "executing instructions from each instruction stream received at the execution unit in a multistage pipeline such that, at a given time, the multistage pipeline includes instructions from different ones of the instruction streams in different stages of the multistage pipeline" of claim 1 (as amended).

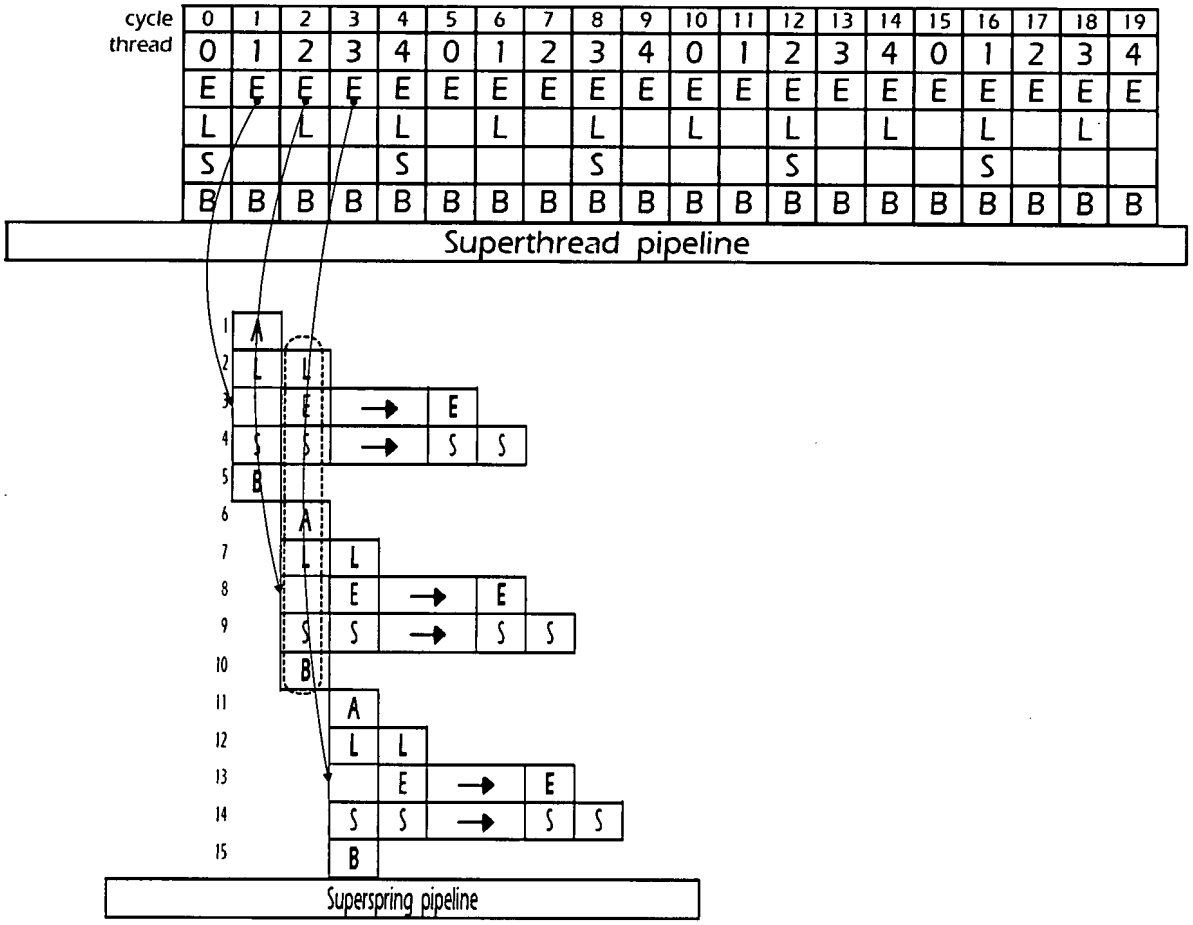
24. A second example pipeline organization is with respect to a Zeus Superspring pipeline, in conjunction with a Zeus Superthread pipeline.

(a) An example Zeus Superspring pipeline is illustrated in a figure on p. 52 of the Zeus manual, and reproduced with annotation directly below:



As one of ordinary skill in the art would have understood based on the foregoing figure and the relevant description provided in the Zeus manual, the foregoing figure illustrates one instruction of each type being issued in a single cycle, with flexible extension to service load operations. The illustration is an example of "a multistage pipeline" operating. Instructions one through five, as annotated, all issue on a first cycle. Instructions six through ten, as annotated, issue on a second cycle. Instructions 11-15, as annotated, issue on a third cycle. Note that, for instance, in the second cycle, the sixth through tenth instructions are in a first stage of the pipeline, while the first through fifth instructions (issued in the first cycle) are in a second stage of the pipeline. Thus on at least the second cycle instructions are in two different stages of the pipeline.

(b) One of ordinary skill in the art would have understood the Zeus Superspring pipeline stages as operated in accordance with the Zeus Superthread pipeline as the following shows. Consider the following figure with annotations superimposed on the foregoing Zeus Superthread and Zeus Superspring pipeline illustrations:



On the first cycle, the first thread of the Zeus Superthread pipeline issues a first group of one or more instructions that use, for instance, the E and B resources. The annotation shows the first group of instructions as it would be processed by the Zeus Superstring pipeline when operated in accordance with the Zeus Superthread pipeline. Similar annotations illustrate the second and third threads issuing respective second and third groups of instructions. Note that, for instance, in the encircled second cycle of the Zeus Superstring stages, the first and the second groups of instructions are in respective ones of the Zeus Superstring pipeline stages, while the instructions of the first and second groups of instructions are respectively from the first and the second threads. Thus on at least the second cycle, the Zeus Superstring pipeline is an example of "[a] multistage pipeline [that] includes instructions from different ones of the instruction streams in different stages of the multistage pipeline".

(c) Thus the second example shows how the disclosure of the '10/757,515 patent application describes the "executing instructions from each instruction stream received at the execution unit in a multistage pipeline such that, at a given time, the multistage pipeline includes instructions from different ones of the instruction streams in different stages of the multistage pipeline" of claim 1 (as amended).

25. A third example pipeline organization is with respect to operation of a Zeus Superthread pipeline.

(a) "A Zeus Superthread pipeline, with 5 simultaneous threads of execution, permits simple operations, such as register-to-register add (G.ADD), to take 5 cycles to complete, allowing for an extremely deeply pipelined implementation." (from p. 53 of the Zeus manual). Completing an operation in "5 cycles" in a "pipelined implementation" implies "a multistage pipeline" as recited in claim 1. Processing of a given instruction would begin on the first of the five cycles, continue on the second of the five cycles, and so forth, and complete on the fifth cycle. Processing of the given instruction would progress through stages of the Superthread pipeline: processing on the first cycle would be in a first stage of the pipeline, processing on the second cycle would be in a second stage of the pipeline, and so forth, with processing on the fifth cycle in a fifth stage of the pipeline.

(b) Consider operation of the foregoing Zeus Superthread multistage pipeline. During a first cycle, the Superthread pipeline begins processing a first instruction from a first thread. The processing (of the first instruction) during the first cycle is within the first stage of the pipeline.

(c) During a second cycle that immediately follows the first cycle, the Superthread pipeline continues to process the first instruction from the first thread. The processing of the first instruction during the second cycle is within the second stage of the pipeline.

(d) Also during the second cycle, the Superthread pipeline begins processing a second instruction from a second thread. The processing of the second instruction during the second cycle is within the first stage of the pipeline.

(e) Thus during the second cycle, the Superthread pipeline is processing an instruction from the first thread (the first instruction) in the second stage of the pipeline, while also processing an instruction from the second thread (the second instruction) in the first stage of the pipeline. Therefore, instructions from "different" streams (the first and the second streams) are in "different stages" (the first and the second stages) of the pipeline at "a given time".

(f) The foregoing description of parallel processing of instructions from a plurality of instruction streams, in a plurality of stages of the Zeus Superthread (multistage) pipeline, is a specific example of "executing instructions from each instruction stream received at the execution unit in a multistage pipeline such that, at a given

time, the multistage pipeline includes instructions from different ones of the instruction streams in different stages of the multistage pipeline" as recited in claim 1 (as amended).

26. Thus the 10/757,515 patent application disclosure, including portions of the Zeus manual, has at least three examples of "executing instructions from each instruction stream received at the execution unit in a multistage pipeline such that, at a given time, the multistage pipeline includes instructions from different ones of the instruction streams in different stages of the multistage pipeline". Therefore the 10/757,515 patent application disclosure has sufficient detail such that a person of ordinary skill in the art could reasonably conclude that the inventors had possession of the "executing instructions from each instruction stream received at the execution unit in a multistage pipeline such that, at a given time, the multistage pipeline includes instructions from different ones of the instruction streams in different stages of the multistage pipeline" of claim 1 (as amended), and that a person of ordinary skill in the art would have been enabled to make and use the claimed invention without undue experimentation.

Summary and Closing:

27. The 10/757,515 patent application, including the Zeus appendix, provides sufficient information in sufficient detail describing the claimed invention that one of ordinary skill in the art would reasonably conclude that the inventors had possession of the claimed invention at the time of filing the 10/757,515 patent application Further, The 10/757,515 patent application, including the Zeus appendix, provides sufficient information regarding the subject matter of the claims to enable one of ordinary skill in the pertinent art to make and use the claimed invention without undue experimentation. Therefore, I believe that the 10/757,515 patent application, including the Zeus appendix, does provide adequate written description and enablement as required by 35 USC § 112 for the "executing instructions from each instruction stream received at the execution unit in a multistage pipeline such that, at a given time, the multistage pipeline includes instructions from different ones of the instruction streams in different stages of the multistage pipeline" of claim 1 (as amended).

28. I have had no communication with any of the inventors of the 10/757,515 patent application (Craig Hansen and John Moussouris) relating to any material in this declaration.

29. I have been hired as a consultant in connection with procedures before the United States Patent and Trademark Office (USPTO) regarding patents and patent applications assigned to Microunity Systems Engineering, Inc., including the media processor patent application. I am being compensated for my services at the rate of $325/hour. Other than acting as a consultant in connection with procedures before the USPTO, I have no interest or connection with Microunity Systems Engineering, Inc.

30. During my evaluation of the media processor patent application, I have been impressed by the thoroughness and overall high-quality of the Zeus manual. The manual provides clear and unambiguous descriptions of media processing systems and is thorough and

well-written. The manual provides comprehensive descriptions of instructions in complete architectural detail. The information in the manual would have been readily understood and easily accessible to software engineers coding the media processing systems, and hardware engineers implementing microprocessors for use in the media processing systems, and that is exactly what an architecture reference manual should be. This is not surprising, since the 10/757,515 patent application includes an architecture manual that is intended to enable hardware engineers to do exactly that – design, build, and implement a media processor that would include circuitry for "executing instructions from each instruction stream received at the execution unit in a multistage pipeline such that, at a given time, the multistage pipeline includes instructions from different ones of the instruction streams in different stages of the multistage pipeline" as described in the Zeus architecture manual.

31. I hereby declare that all statements made herein are of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issuing therefrom.

Date: May 25, 2007

Korbin Van Dyke: K. Van Dyke

Address: 3343 Little Valley Rd.
Sunol, CA 94586

Korbin S. Van Dyke

3343 Little Valley Road
Sunol, CA 94586
(925) 862-0665
mail: Korbin@KorbinVanDyke.com

Personal Statement

I am an expert in Microprocessor Logic Design, VLSI Systems, and related intellectual property. Throughout more than twenty years of involvement in the specification, implementation, and debugging of complex computer architecture systems, I've enjoyed the daily challenge of learning, focusing on details, and providing working systems to end customers. In August, 2002, I became a registered US patent agent and intellectual property consultant. I now enjoy aiding clients and their counsel in understanding these technologies, and protecting their intellectual property investments in these areas.

Experience Summary

- 04/2004-present
  - Owner/Operator, Korbin S Van Dyke Consulting, office in Sunol, California

I am assisting clients in understanding technical aspects of complex intellectual property issues with emphasis on complex digital systems corresponding to my technical expertise. I have acted on behalf of plaintiffs and defendants in various intellectual property disputes.

Acting as a technical expert on a successful licensing project, I analyzed a collection of more than 100 computer microarchitecture patents and related prior art. I identified potentially infringing products based on publicly available information, and produced a collection of associated claim charts. In addition, I assisted the client in drafting further claims targeting other potentially infringing products.

- 01/2002-present
  - Senior Member Technical Staff and Sub-Contractor, PatentVentures, offices in California and Texas

I am authoring complex patent applications, analyzing existing intellectual property, and expanding the scope of client patent coverage. These contributions are in the areas of complex VLSI system design, leveraging my technical expertise relevant to these systems.

- 10/2001-12/2001
  - Architect, Clearwater Networks (previously XStream Logic), Los Gatos, CA

I was the senior technical contributor in the architecture group. I was responsible for defining, documenting, and championing the architecture of future network processor products.

- 06/2000-09/2001
  - Director, VLSI Development, XStream Logic, Los Gatos, CA

I was responsible for building and leading the VLSI team developing a complex network processor. I owned the overall chip implementation and debug schedule, resource management decisions, and was also the senior technical contributor. The scope of work included front end design, verification, back end design, and debug. I built the team from ten to 35 members, and increased the skill set from front end logic design and EDA to include circuit, micro-architecture, and verification. In addition, I lead a "virtual ops" group until the operations team came on board.

- 06/1999-05/2000
  - Senior Manager, Architecture, ATI Research Silicon Valley, Santa Clara, CA

Previous positions (at ATI):
  - Manager, ATI Research Silicon Valley, Santa Clara, CA

I was responsible for the top-level CPU architectural definition and partitioning, working directly with a technical writer to specify the architecture unambiguously. This requires a micro-code programmer's model negotiated between the micro-code and hardware micro-architecture implementers and logic designers. I lead the micro-code development and performance analysis teams. I also directly managed the place and route group.

I was responsible for leading a cross-functional team to design in complete X86 compatibility. This requires investigations into ambiguous or complex cases, specifying the architecture of the solution (hardware and/or software), and coordinating the complex hardware/software solutions.

I worked with contributors across the CPU to encourage and facilitate filing patent applications covering the inventions at architectural and implementation levels.

I led a small team designing the pipeline control section of a super-scalar in-order x86 compatible processor. The team was responsible for unit definition and micro-architecture, block partitioning, RTL coding, synthesis, custom macro-specification, timing analysis and improvement, cell placement, and cell routing. The team was charged with meeting area, timing, and bug count goals. I coordinated the activities of the group and negotiated interfaces with the other three units of the processor: memory, instruction fetch/conversion, and datapaths, in addition to providing specific technical guidance and solving problems.

- 05/1996-5/1999
  - Member Technical Staff, Architect, Chromatic Research, Sunnyvale, CA (merged with ATI Technology Nov 1998)

I drove closure of the final details of the programmer's model of a complex instruction set architecture device with multiple processors. Each processor is super-scalar (in order), dual instruction set capable (industry standard X86 plus proprietary native), with hardware

optimizations to support binary translation and media operations (MPEG decoding and encoding, decryption, and 3D lighting and geometry calculations).

I was a key contributor to the team refining the product definition, system and CPU level partitioning, CPU ISA, and internal CPU micro-architecture. I personally connected logic and circuit designers with media programmers via negotiation and analysis to enable a high performance well thought out and cost effective total solution.

I examined external intellectual property to identify exposures and workarounds. I contributed directly to the creation of internal intellectual property: multiple ISA execution, hardware optimization for binary translation, video decoding ISA optimization, X86 compatible ISA implementation techniques, standard PC sub-system virtualization mechanisms, and compatible segmentation and paging methods.

- 10/1987-05/1996
  - Senior Manager, AMD (formerly Nexgen), Milpitas, CA

Previous positions (at Nexgen):
- Director Processor Development
- Project manger Architecture Development
- Senior Member Technical Staff
- Member Technical Staff

I investigated possible micro-architectures for the eighth generation x86 ISA implementation, including: overall CPU performance (using an in-house performance simulation tool), expected cycle time (based on logic design of critical paths), and required process technology.

I was a key contributor toward the first Nexgen product (Nx586) – a super-scalar, out of order, fifth generation implementation of the x86 ISA. This product briefly shipped to customers.

My individual contributions included: pipeline control logic design, branch prediction and multiple stream instruction fetch micro-architecture and logic design, lab debug, timing bug identification and resolution, performance analysis and improvement, and overall design correctness.

Management responsibilities included: architectural verification, lab debug, and eventually the entire implementation of the CPU in groups from 4 to 20 people.

- 6/1982-10/1987
  - VLSI Systems Engineer, VLSI Technology, San Jose, CA

I led a small team as an active manager in the design, implementation, and prototype evaluation of a programmable digital signal processor. The project began with the definition of the instruction set architecture, and culminated in a working modem demonstration constructed from the first silicon.

I specified and designed the memory interface for a multi-mode display interface chip (first silicon was used for product demonstration), reverse engineered a small CMOS standard part chip, and supported a telecommunications chip set customer design project.

- 6/1981-6/1982
  - Research Assistant, UC Berkeley, Berkeley, CA

I assisted in the NMOS logic specification, logic verification, functional simulation, circuit design, and layout verification of an implementation of a Reduced Instruction Set Computer (RISC-I).

Education

6/1982 MS, EECS	UC Berkeley, Berkeley, CA	GPA: 4.0/4.0
6/1980 BS, EECS	UC Berkeley, Berkeley, CA	GPA: 3.9/4.0

Activities

I have been inducted into: Phi Beta Kapa, Tau Beta Pi , and Eta Kappa Nu
I am a member of IEEE (since 1978)
I am a member of ACM (since 1983)

Patents

I am a co-inventor on more than 50 patents. These patents relate to a variety of subjects, including: multiple ISA execution, hardware optimization for binary translation, video decoding ISA optimization, compatible ISA implementation techniques, speculative instruction execution, branch prediction, compatible segmentation and paging, standard PC sub-system virtualization, address generation, and logarithmic calculation.

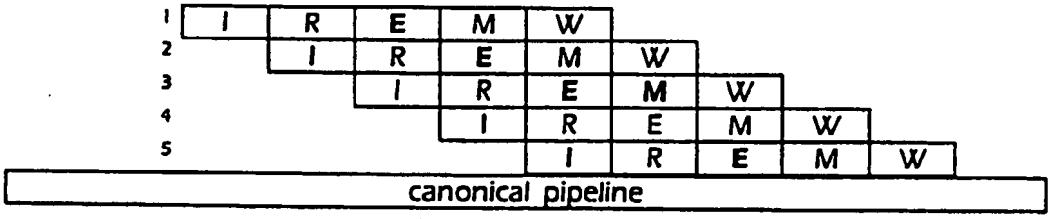
Pipeline Organization

Zeus performs all instructions *as if* executed one-by-one, in-order, with precise exceptions always available. Consequently, code that ignores the subsequent discussion of Zeus pipeline implementations will still perform correctly. However, the highest performance of the Zeus processor is achieved only by matching the ordering of instructions to the characteristics of the pipeline. In the following discussion, the general characteristics of all Zeus implementations precede discussion of specific choices for specific implementations.

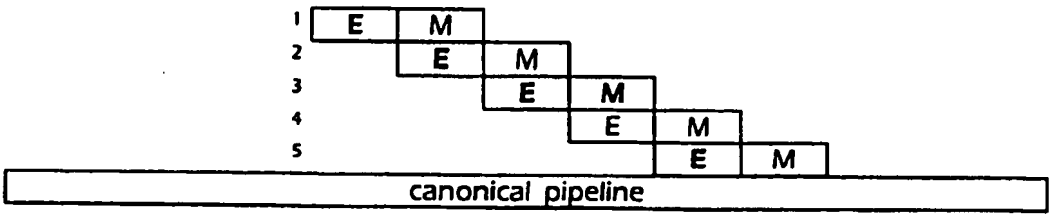
Classical Pipeline Structures

Pipelining in general refers to hardware structures that overlap various stages of execution of a series of instructions so that the time required to perform the series of instructions is less than the sum of the times required to perform each of the instructions separately. Additionally, pipelines carry to connotation of a collection of hardware structures which have a simple ordering and where each structure performs a specialized function.

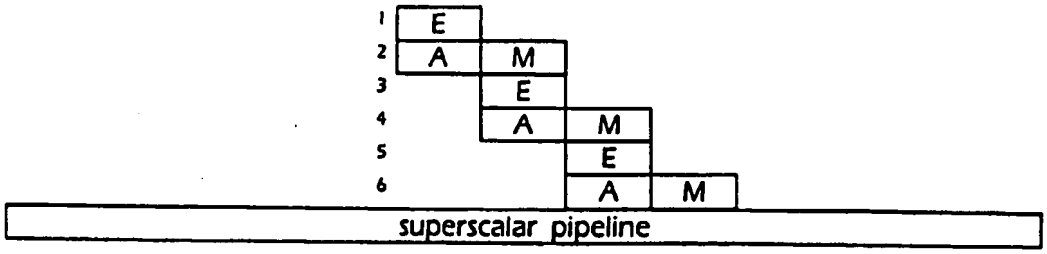
The diagram below shows the timing of what has become a canonical pipeline structure for a simple RISC processor, with time on the horizontal axis increasing to the right, and successive instructions on the vertical axis going downward. The stages I, R, E, M, and W refer to units which perform instruction fetch, register file fetch, execution, data memory fetch, and register file write. The stages are aligned so that the result of the execution of an instruction may be used as the source of the execution of an immediately following instruction, as seen by the fact that the end of an E stage (bold in line 1) lines up with the beginning of the E stage (bold in line 2) immediately below. Also, it can be seen that the result of a load operation executing in stages E and M (bold in line 3) is not available in the immediately following instruction (line 4), but may be used two cycles later (line 5); this is the cause of the load delay slot seen on some RISC processors.



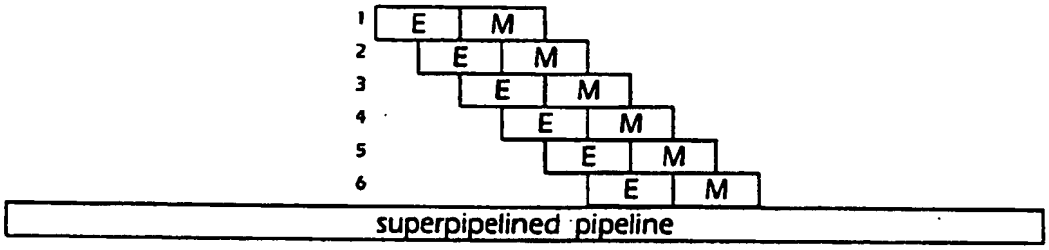
In the diagrams below, we simplify the diagrams somewhat by eliminating the pipe stages for instruction fetch, register file fetch, and register file write, which can be understood to precede and follow the portions of the pipelines diagrammed. The diagram above is shown again in this new format, showing that the canonical pipeline has very little overlap of the actual execution of instructions.



A superscalar pipeline is one capable of simultaneously issuing two or more instructions which are independent, in that they can be executed in either order and separately, producing the same result as if they were executed serially. The diagram below shows a two-way superscalar processor, where one instruction may be a register-to-register operation (using stage E) and the other may be a register-to-register operation (using stage A) or a memory load or store (using stages A and M).



A superpipelined pipeline is one capable is issuing simple instructions frequently enough that the result of a simple instruction must be independent of the immediately following one or more instructions. The diagram below shows a two-cycle superpipelined implementation:



In the diagrams below, pipeline stages are labelled with the type of instruction that may be performed by that stage. The posititon of the stage further identifies the function of that stage, as for example a load operation may require several L stages to complete the instruction.

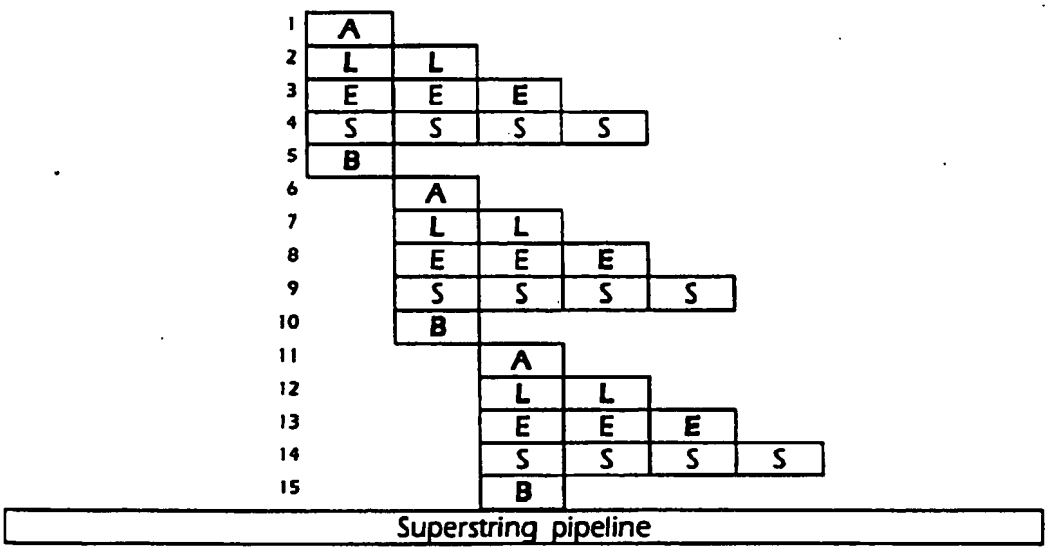
Superstring Pipeline

Zeus architecture provides for implementations designed to fetch and execute several instructions in each clock cycle. For a particular ordering of instruction types, one instruction of each type may be issued in a single clock cycle. The ordering required is A, L, E, S, B; in other words, a register-to-register address calculation, a memory load, a register-to-register data calculation, a memory store, and a branch. Because of the organization of the pipeline, each of these instructions may be serially dependent. Instructions of type E include the fixed-point execute-phase instructions as well as floating-point and digital signal processing instructions. We call this form of pipeline organization "superstring,"[4] because of the ability to issue a string of dependent instructions in a single clock cycle, as distinguished

[4]Readers with a background in theoretical physics may have seen this term in an other, unrelated, context.

from superscalar or superpipelined organizations, which can only issue sets of independent instructions.

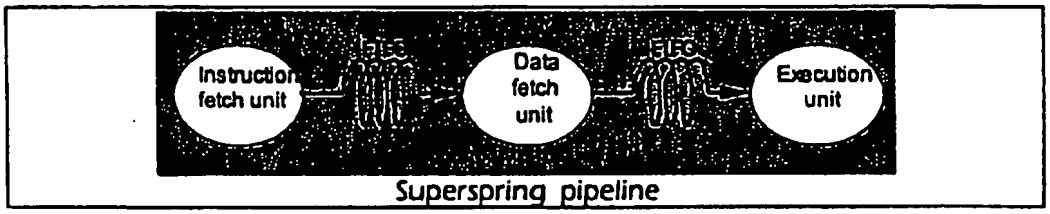
These instructions take from one to four cycles of latency to execute, and a branch prediction mechanism is used to keep the pipeline filled. The diagram below shows a box for the interval between issue of each instruction and the completion. Bold letters mark the critical latency paths of the instructions, that is, the periods between the required availability of the source registers and the earliest availability of the result registers. The A-L critical latency path is a special case, in which the result of the A instruction may be used as the base register of the L instruction without penalty. E instructions may require additional cycles of latency for certain operations, such as fixed-point multiply and divide, floating-point and digital signal processing operations.



Superspring Pipeline

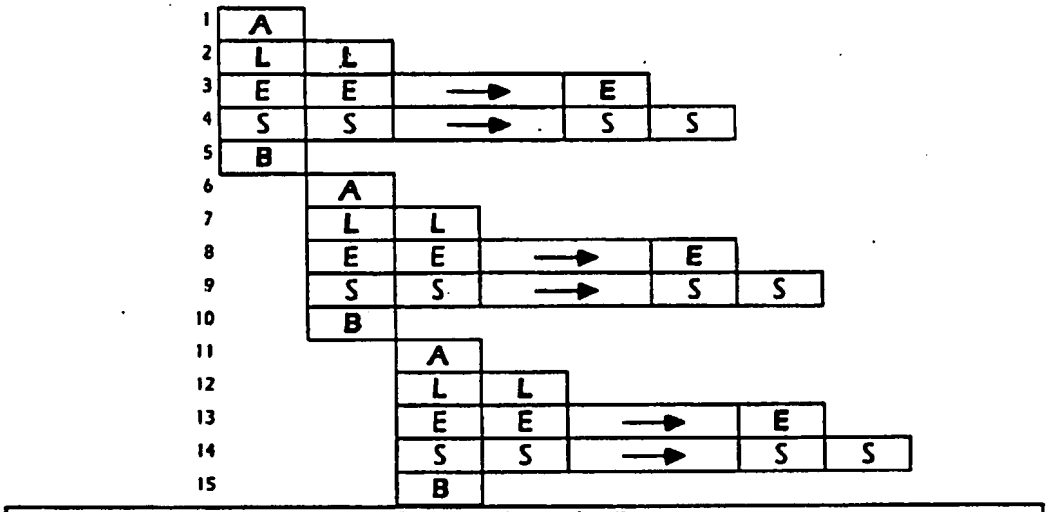
Zeus architecture provides an additional refinement to the organization defined above, in which the time permitted by the pipeline to service load operations may be flexibly extended. Thus, the front of the pipeline, in which A, L and B type instructions are handled, is decoupled from the back of the pipeline, in which E, and S type instructions are handled. This decoupling occurs at the point at which the data cache and its backing memory is referenced; similarly, a FIFO that is filled by the instruction fetch unit decouples instruction cache references from the front of the pipeline shown above. The depth of the FIFO structures is implementation-dependent, i.e. not fixed by the architecture.

The diagram below indicates why we call this pipeline organization feature "superspring," an extension of our superstring organization.



Superspring pipeline

With the super-spring organization, the latency of load instructions can be hidden, as execute instructions are deferred until the results of the load are available. Nevertheless, the execution unit still processes instructions in normal order, and provides precise exceptions.



Superspring pipeline

Superthread Pipeline

This technique is not employed in the initial Zeus implementation, though it was present in an earlier prototype implementation.

A difficulty of superpipelining is that dependent operations must be separated by the latency of the pipeline, and for highly pipelined machines, the latency of simple operations can be quite significant. The Zeus "superthread" pipeline provides for very highly pipelined implementations by alternating execution of two or more independent threads. In this context, a thread is the state required to maintain an independent execution; the architectural state required is that of the register file contents, program counter, privilege level, local TB, and when required, exception status. Ensuring that only one thread may handle an exception at one time may minimize the latter state, exception status. In order to ensure that all threads make reasonable forward progress, several of the machine resources must be scheduled fairly.

An example of a resource that is critical that it be fairly shared is the data memory/cache subsystem. In a prototype implementation, Zeus is able to perform a load operation only on every second cycle, and a store operation only on every fourth cycle. Zeus schedules these fixed timing resources fairly by using a round-robin schedule for a number of threads that is relatively prime to the resource reuse rates. For this implementation, five simultaneous threads of execution ensure that resources which may be used every two or four cycles are fairly shared by allowing the instructions which use those resources to be issued only on every second or fourth issue slot for that thread.

In the diagram below, the thread number which issues an instruction is indicated on each clock cycle, and below it, a list of which functional units may be used by that instruction. The diagram repeats every 20 cycles, so cycle 20 is similar to cycle 0, cycle 21 is similar to cycle 1, etc. This schedule ensures that no resource conflict occur between threads for these resources. Thread 0 may issue an E, L, S or B on cycle 0, but on its next opportunity, cycle 5, may only issue E or B, and on cycle 10 may issue E, L or B, and on cycle 15, may issue E or B.

cycle	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
thread	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4	0	1	2	3	4
	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
	L		L		L		L		L		L		L		L		L		L	
	S				S				S				S				S			
	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B

Superthread pipeline

When seen from the perspective of an individual thread, the resource use diagram looks similar to that of the collection. Thus an individual thread may use the load unit every two instructions, and the store unit every four instructions.

cycle	0	5	10	15	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95
thread	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E	E
	L		L		L		L		L		L		L		L		L		L	
	S				S				S				S				S			
	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B	B

Superthread pipeline

A Zeus Superthread pipeline, with 5 simultaneous threads of execution, permits simple operations, such as register-to-register add (G.ADD), to take 5 cycles to complete, allowing for an extremely deeply pipelined implementation.

Simultaneous Multithreading

The intial Zeus implementation performs simultaneous multithreading among 4 threads. Each of the 4 threads share a common memory system, a common T unit. Pairs of threads share two G units, one X unit, and one E unit. Each thread individually has two A units. A fair allocation scheme balances access to the shared resources by the four threads.

Group Add

These operations take operands from two registers, perform operations on partitions of bits in the operands, and place the concatenated results in a third register.

Operation codes

G.ADD.8	Group add bytes
G.ADD.16	Group add doublets
G.ADD.32	Group add quadlets
G.ADD.64	Group add octlets
G.ADD.128	Group add hexlet
G.ADD.L.8	Group add limit signed bytes
G.ADD.L.16	Group add limit signed doublets
G.ADD.L.32	Group add limit signed quadlets
G.ADD.L.64	Group add limit signed octlets
G.ADD.L.128	Group add limit signed hexlet
G.ADD.L.U.8	Group add limit unsigned bytes
G.ADD.L.U.16	Group add limit unsigned doublets
G.ADD.L.U.32	Group add limit unsigned quadlets
G.ADD.L.U.64	Group add limit unsigned octlets
G.ADD.L.U.128	Group add limit unsigned hexlet
G.ADD.8.O	Group add signed bytes check overflow
G.ADD.16.O	Group add signed doublets check overflow
G.ADD.32.O	Group add signed quadlets check overflow
G.ADD.64.O	Group add signed octlets check overflow
G.ADD.128.O	Group add signed hexlet check overflow
G.ADD.U.8.O	Group add unsigned bytes check overflow
G.ADD.U.16.O	Group add unsigned doublets check overflow
G.ADD.U.32.O	Group add unsigned quadlets check overflow
G.ADD.U.64.O	Group add unsigned octlets check overflow
G.ADD.U.128.O	Group add unsigned hexlet check overflow

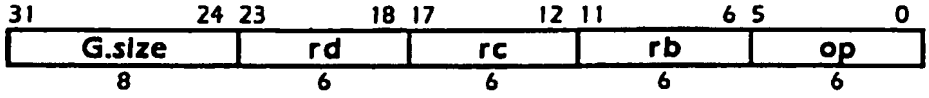
Redundancies

G.ADD.size rd=rc,rc	⇔	G.SHL.I.size rd=rc,1
G.ADD.size.O rd=rc,rc	⇔	G.SHL.I.size.O rd=rc,1
G.ADD.U.size.O rd=rc,rc	⇔	G.SHL.I.U.size.O rd=rc,1

Format

G.op.sizerd=rc,rb

rd=gopsize(rc,rb)



Description

The contents of registers rc and rb are partitioned into groups of operands of the size specified and added, and if specified, checked for overflow or limited, yielding a group of results, each of which is the size specified. The group of results is catenated and placed in register rd.

Definition

```
def Group(op,size,rd,rc,rb)
    c ← RegRead(rc, 128)
    b ← RegRead(rb, 128)
    case op of
        GADD:
            for i ← 0 to 128-size by size
                a[i+size-1..i] ← c[i+size-1..i] + b[i+size-1..i]
            endfor
        GADD.L:
            for i ← 0 to 128-size by size
                t ← (c[i+size-1] || c[i+size-1..i]) + (b[i+size-1] || b[i+size-1..i])
                a[i+size-1..i] ← (t[size] ≠ t[size-1]) ? (t[size] || t[size-1]^(size-1)) : t[size-1..0]
            endfor
        GADD.LU:
            for i ← 0 to 128-size by size
                t ← (0[1] || c[i+size-1..i]) + (0[1] || b[i+size-1..i])
                a[i+size-1..i] ← (t[size] ≠ 0) ? (1^size) : t[size-1..0]
            endfor
        GADD.O:
            for i ← 0 to 128-size by size
                t ← (c[i+size-1] || c[i+size-1..i]) + (b[i+size-1] || b[i+size-1..i])
                if t[size] ≠ t[size-1] then
                    raise FixedPointArithmetic
                endif
                a[i+size-1..i] ← t[size-1..0]
            endfor
        GADD.U.O:
            for i ← 0 to 128-size by size
                t ← (0[1] || c[i+size-1..i]) + (0[1] || b[i+size-1..i])
                if t[size] ≠ 0 then
                    raise FixedPointArithmetic
                endif
                a[i+size-1..i] ← t[size-1..0]
```

```
            endfor
    endcase
    RegWrite(rd, 128, a)
enddef
```

Exceptions

Fixed-point arithmetic

Ensemble Floating-point

These operations take two values from registers, perform a group of floating-point arithmetic operations on partitions of bits in the operands, and place the catenated results in a register.

Operation codes

E.ADD.F.16	Ensemble add floating-point half
E.ADD.F.16.C	Ensemble add floating-point half ceiling
E.ADD.F.16.F	Ensemble add floating-point half floor
E.ADD.F.16.N	Ensemble add floating-point half nearest
E.ADD.F.16.X	Ensemble add floating-point half exact
E.ADD.F.16.Z	Ensemble add floating-point half zero
E.ADD.F.32	Ensemble add floating-point single
E.ADD.F.32.C	Ensemble add floating-point single ceiling
E.ADD.F.32.F	Ensemble add floating-point single floor
E.ADD.F.32.N	Ensemble add floating-point single nearest
E.ADD.F.32.X	Ensemble add floating-point single exact
E.ADD.F.32.Z	Ensemble add floating-point single zero
E.ADD.F.64	Ensemble add floating-point double
E.ADD.F.64.C	Ensemble add floating-point double ceiling
E.ADD.F.64.F	Ensemble add floating-point double floor
E.ADD.F.64.N	Ensemble add floating-point double nearest
E.ADD.F.64.X	Ensemble add floating-point double exact
E.ADD.F.64.Z	Ensemble add floating-point double zero
E.ADD.F.128	Ensemble add floating-point quad
E.ADD.F.128.C	Ensemble add floating-point quad ceiling
E.ADD.F.128.F	Ensemble add floating-point quad floor
E.ADD.F.128.N	Ensemble add floating-point quad nearest
E.ADD.F.128.X	Ensemble add floating-point quad exact
E.ADD.F.128.Z	Ensemble add floating-point quad zero
E.DIV.F.16	Ensemble divide floating-point half
E.DIV.F.16.C	Ensemble divide floating-point half ceiling
E.DIV.F.16.F	Ensemble divide floating-point half floor
E.DIV.F.16.N	Ensemble divide floating-point half nearest
E.DIV.F.16.X	Ensemble divide floating-point half exact
E.DIV.F.16.Z	Ensemble divide floating-point half zero
E.DIV.F.32	Ensemble divide floating-point single
E.DIV.F.32.C	Ensemble divide floating-point single ceiling
E.DIV.F.32.F	Ensemble divide floating-point single floor
E.DIV.F.32.N	Ensemble divide floating-point single nearest
E.DIV.F.32.X	Ensemble divide floating-point single exact
E.DIV.F.32.Z	Ensemble divide floating-point single zero
E.DIV.F.64	Ensemble divide floating-point double
E.DIV.F.64.C	Ensemble divide floating-point double ceiling

E.DIV.F.64.F	Ensemble divide floating-point double floor
E.DIV.F.64.N	Ensemble divide floating-point double nearest
E.DIV.F.64.X	Ensemble divide floating-point double exact
E.DIV.F.64.Z	Ensemble divide floating-point double zero
E.DIV.F.128	Ensemble divide floating-point quad
E.DIV.F.128.C	Ensemble divide floating-point quad ceiling
E.DIV.F.128.F	Ensemble divide floating-point quad floor
E.DIV.F.128.N	Ensemble divide floating-point quad nearest
E.DIV.F.128.X	Ensemble divide floating-point quad exact
E.DIV.F.128.Z	Ensemble divide floating-point quad zero
E.MUL.C.F.16	Ensemble multiply complex floating-point half
E.MUL.C.F.32	Ensemble multiply complex floating-point single
E.MUL.C.F.64	Ensemble multiply complex floating-point double
E.MUL.F.16	Ensemble multiply floating-point half
E.MUL.F.16.C	Ensemble multiply floating-point half ceiling
E.MUL.F.16.F	Ensemble multiply floating-point half floor
E.MUL.F.16.N	Ensemble multiply floating-point half nearest
E.MUL.F.16.X	Ensemble multiply floating-point half exact
E.MUL.F.16.Z	Ensemble multiply floating-point half zero
E.MUL.F.32	Ensemble multiply floating-point single
E.MUL.F.32.C	Ensemble multiply floating-point single ceiling
E.MUL.F.32.F	Ensemble multiply floating-point single floor
E.MUL.F.32.N	Ensemble multiply floating-point single nearest
E.MUL.F.32.X	Ensemble multiply floating-point single exact
E.MUL.F.32.Z	Ensemble multiply floating-point single zero
E.MUL.F.64	Ensemble multiply floating-point double
E.MUL.F.64.C	Ensemble multiply floating-point double ceiling
E.MUL.F.64.F	Ensemble multiply floating-point double floor
E.MUL.F.64.N	Ensemble multiply floating-point double nearest
E.MUL.F.64.X	Ensemble multiply floating-point double exact
E.MUL.F.64.Z	Ensemble multiply floating-point double zero
E.MUL.F.128	Ensemble multiply floating-point quad
E.MUL.F.128.C	Ensemble multiply floating-point quad ceiling
E.MUL.F.128.F	Ensemble multiply floating-point quad floor
E.MUL.F.128.N	Ensemble multiply floating-point quad nearest
E.MUL.F.128.X	Ensemble multiply floating-point quad exact
E.MUL.F.128.Z	Ensemble multiply floating-point quad zero

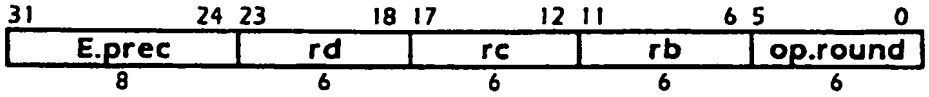
Selection

class	op	prec				round/trap
add	EADDF	16	32	64	128	NONE C F N X Z
divide	EDIVF	16	32	64	128	NONE C F N X Z
multiply	EMULF	16	32	64	128	NONE C F N X Z
complex multiply	EMUL.CF	16	32	64		NONE

Format

E.op.prec.round rd=rc,rb

rd=eopprecround(rc,rb)



Description

The contents of registers ra and rb are combined using the specified floating-point operation. The result is placed in register rc. The operation is rounded using the specified rounding option or using round-to-nearest if not specified. If a rounding option is specified, the operation raises a floating-point exception if a floating-point invalid operation, divide by zero, overflow, or underflow occurs, or when specified, if the result is inexact. If a rounding option is not specified, floating-point exceptions are not raised, and are handled according to the default rules of IEEE 754.

Definition

```
def mul(size,v,i,w,j) as
    mul ← fmul(F(size,v_size-1+i..i),F(size,w_size-1+j..j))
enddef

def EnsembleFloatingPoint(op,prec,round,ra,rb,rc) as
    c ← RegRead(rc, 128)
    b ← RegRead(rb, 128)
    for i ← 0 to 128-prec by prec
        ci ← F(prec,c_i+prec-1..i)
        bi ← F(prec,b_i+prec-1..i)
        case op of
            E.ADD.F:
                ai ← faddr(ci,bi,round)
            E.MUL.F:
                ai ← fmul(ci,bi)
            E.MUL.C.F:
                if (i and prec) then
                    ai ← fadd(mul(prec,c,i,b,i-prec), mul(prec,c,i-prec,b,i))
                else
                    ai ← fsub(mul(prec,c,i,b,i), mul(prec,c,i+prec,b,i+prec))
                endif
            E.DIV.F.:
                ai ← fdiv(ci,bi)
        endcase
        a_i+prec-1..i ← PackF(prec, ai, round)
    endfor
    RegWrite(rd, 128, a)
enddef
```

Exceptions

Floating-point arithmetic